U-Boot for MPL SBC

User Manual

U-Boot Bootloader for MPL Single Board Computer

The U-Boot is the bootloader of the MPL Single Board Computers PIP405, MIP405, MIP405T and the VCMA9. Its main purpose is to initialize the hardware, set-up the SDRAM and boot a kernel image. But it acts also as monitor and has build in a lot of useful functions such as memory dump, memory write etc. It allows to update the U-Boot without having to boot an operating system.

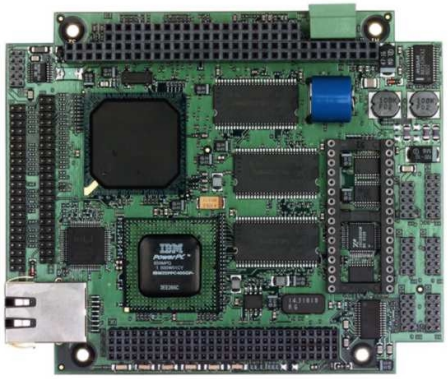
With the permanently storable environment variables U-Boot is highly configurable, so the boot device, the initial console interface, boot parameters etc. are configurable.

The U-Boot is an open source project. The source is available on http://u-boot.sourceforge.net/ . Please refer to its documentation for further information.

The U-Boot is closely related to the Linux and special provision has been made to support booting of Linux kernel images.

Features

- Power On Self Test (POST)
- Various boot device support such as:
  - Ethernet
  - USB
  - IDE Devices
  - SCSI Devices
  - ATAPI Devices
  - Floppy Disk
- Easy U-Boot update
- Highly configurable via permanently storable environment variables.
- Initial console configurable as serial line or VGA/Keyboard.
- Supports booting of compressed (gzip) kernel images.



© 2003 by MPL AG

Table of contents

1. INTRODUCTION 4

1.1 About this manual 4
1.2 Conventions 4

2. GENERAL INFORMATION AND SPECIFICATIONS 5

2.1 Board depending Features 5
2.2 Common Features 5

3. USING U-BOOT 6

3.1 Start screen 6
3.1.1 Information of the start-up screen 6
3.2 U-Boot Command Interface 7
3.2.1 Help system 7
3.2.2 Command modifiers 8
3.3 U-Boot Commands 9
3.3.1 base - print or set address offset 9
3.3.2 bdinfo - print Board Info structure 9
3.3.3 bootd - boot default, i.e run 'bootcmd' 10
3.3.4 bootm - boot application image from memory 10
3.3.5 bootp - boot image via network using BOOTP/TFTP protocol 11
3.3.6 cmp - memory compare 11
3.3.7 coninfo - print console devices and information 11
3.3.8 cp - memory copy 11
3.3.9 crc32 - checksum calculation 12
3.3.10 date - get/set/reset date & time 12
3.3.11 dhcp - invoke DHCP client to obtain IP/boot params 12
3.3.12 diskboot- boot from IDE device 12
3.3.13 docboot - boot from DOC device 13
3.3.14 doc - Disk-On-Chip sub-system 13
3.3.15 erase - erase FLASH memory 13
3.3.16 fdcboot - boot from floppy device 14
3.3.17 flinfo - print FLASH memory information 14
3.3.18 fsinfo - print information about filesystems 14
3.3.19 fsload - load binary file from a filesystem image 14
3.3.20 ide - IDE sub-system 15
3.3.21 isdram - print SDRAM configuration information 15
3.3.22 iminfo - print header information for application image 16
3.3.23 irqinfo - print information about IRQs 16
3.3.24 loadb - load binary file over serial line (kermit mode) 16
3.3.25 md - memory display 17
3.3.26 mm - memory modify (auto-incrementing) 17
3.3.27 mtest - simple RAM test 17
3.3.28 mw - memory write (fill) 18
3.3.29 nand - NAND sub-system 18
3.3.30 nboot - boot from NAND device 19
3.3.31 nm - memory modify (constant address) 19
3.3.32 pci - list and access PCI Configuration Space 19
3.3.33 printenv- print environment variables 20
3.3.34 protect - enable or disable FLASH write protection 20
3.3.35 rarpboot- boot image via network using RARP/TFTP protocol 21
3.3.36 reset - Perform RESET of the CPU 21
3.3.37 run - run commands in a environment variable 21
3.3.38 saveenv - save environment variables to persistent storage 22
3.3.39 scsiboot- boot from SCSI device 22
3.3.40 scsi - SCSI sub-system 22
3.3.41 setenv - set environment variables 22
3.3.42 tftpboot- boot image via network using TFTP protocol 23
3.3.43 usb - USB sub-system 23
3.3.44 usbboot - boot from USB device 25

© 2003 by MPL AG

3.3.45 pip405/mip405/vcma9 - Board specific Cmds 25
3.4 U-Boot Environment Variables 26
3.4.1 Board properties Variables 26
3.4.2 Variables controlling the OS boot 26
3.4.3 Variables for U-Boot behavior 26
3.4.4 Network Variables 26
3.4.5 Automatically updated Variables 27
3.5 Procedures 28
3.5.1 Updating the bootloader 28
3.5.1.1 Load the image in the memory 28
3.5.1.2 Program the flash 28
3.5.2 Programming a kernel image into the flash 29
3.5.2.1 Erase the flash 29
3.5.2.2 Load the image into the memory 29
3.5.2.3 Write the image into the flash 30
3.5.2.4 Booting the flashed kernel 30
3.5.3 Console assignment 31
3.5.3.1 Assign alternate console 31
3.5.3.2 Working with an USB Keyboard 32
3.5.4 Creating bootable kernel images 33
3.5.4.1 U-Boot images 33
3.5.4.2 Using mkimage 33
3.5.5 Creating a new U-Boot Image 34

4. SUPPORT INFORMATION 35

4.1 U-Boot project 35
4.2 MPL AG 35

© 2003 by MPL AG

1. Introduction

1.1 About this manual

This manual gives a brief introduction of the U-Boot bootloader. Since the U-Boot is an open source project which might be updated frequently, some information may be not up to date. The purpose of this manual is to provide enough information for using (booting) the MPL Single Board Computers. For more information please refer to the source code and its documentation at http://u-boot.sourceforge.net/.
All hardware related information such as DIP-switch settings, connecting peripherals etc can be found in the appropriate User Manual supplied by MPL AG or your local MPL products supplier. The User Manuals are also available on the internet under www.mpl.ch in PDF format.
The manual is written for technical personnel responsible for integrating the MPL Single Board Computers into their system.

1.2 Conventions

Following text conventions are used in this Manual:

Text Output from U-Boot:	`U-Boot Output`
Command Input:	***`Command Input`***
U-Boot Prompt:	`=>`
Board name (pip405, mip405, vcma9)	`BOARD`
Loadaddress	`LOADADDRESS`

- All numbers in U-Boot commands and parameters are hexadecimal.
- The commands and parameters are case sensitive.
- Size parameters are in units. Default unit is long `.l` (32bit). Use modifier `.b` for bytes (8bit) `.w` for words (16Bit) see chapter 3.2.2 Command modifiers for further information.
- The `LOADADDRESS` is 400000 on the PIP405 and MIP405 and 33000000 on a VCMA9

© 2003 by MPL AG

2. General Information and Specifications

This chapter provides a general overview over the U-Boot and its features. Please note that some of these features (such as SCSI, VGA) are board depended and may not apply to your MPL Single Board Computer.

2.1 Board depending Features

Following table lists the implemented Features of the currently available MPL Single Board Computers:

Features	PIP405	MIP405	MIP405T	VCMA9
Initialization / Power-On Self Test				
SDRAM	√	√	√	√
Ethernet controller	√	√	√	√
Serial lines	√	√	√	√
IDE controller	√	√	√	N/A
USB controller	√	√	N/A	X
SCSI controller	√[1]	N/A	N/A	N/A
Display controller	√[1]	√[2]	√[2]	X
Keyboard controller	√[1]	N/A	N/A	N/A
Boot device support:				
On-board Flash	√	√	√	√
Serial Line	√	√	√	√
Ethernet	√	√	√	√
IDE Device	√	√	√	N/A
SCSI Device	√[1]	N/A	N/A	N/A
ATAPI Device (CD-ROM)	√	√	√	N/A
Floppy Disk	√	N/A	N/A	N/A
USB.	√	√	N/A	X
Bootloader Update:				
Image in the Memory (loadable from serial line, Ethernet etc)	√	√	√	√
MPS (Multi Purpose Socket)	√	√	N/A	√[3]
Miscellaneous:				
RTC support.	√	√	√[4]	√[4]

Legend:

√ Supported
X Currently Not Supported
N/A Not Available

Notes:

1. Only if populated
2. Supports PC104 Plus Cards with C&T69000 Display controller
3. Only with VCMA9-BB
4. Only with external back-up Battery

2.2 Common Features

Configuration:

- Various Options can be set by Environment variables.
- Environment variables are permanently stored in the on-board I2C EEPROM
- Environment variables are protected with a CRC32 checksum

Kernel booting:

- Supports compressed (gzip) kernel images.
- Images are checked with a CRC32 checksum prior of booting.
- Autoboot option.
- Boot arguments can be stored in the environment variables.

Common Features for all

- Monitor Functions

© 2003 by MPL AG

3. Using U-Boot

This section describes the generally use of the U-Boot. Normally the MPL boards use serial line 0 as its console device. The parameters of this serial communication are as follows by default:

- 9600 baud
- 8 bits
- parity none
- 1 stop bit
- no handshake

Use your favorite terminal program (kermit, minicom, ..) to get in contact with U-Boot's command interface.

Please note that you have to use a so called "null modem" or "crossed" cable to get a working communication link.

3.1 Start screen

Depending on the U-Boot version and the board, the start screen looks different. Its main purpose is to provide information of the board and the status of the peripherals. Below is a start screen just for reference:

```
U-Boot 0.3.2 (Jun  4 2003 - 14:07:51)
(c) 2003 by MPL AG Switzerland, MEV-10072-001 released

CPU:   IBM PowerPC 405GP Rev. E at 198 MHz (PLB=99, OPB=33, EBC=24 MHz)
       PCI sync clock at 33 MHz, internal PCI arbiter enabled

Board: PIP405 SN: 500 Boot Config: 0xe
DRAM:  single-sided DIMM 128 MB
FLASH: (Flash Boot)  4 MB
IDE:   Bus 0: ..OK Bus 1: OK
  Device 0: Model: HITACHI_DK239A-48 Firm: 00X8A0A2 Ser#: 448290
            Type: Hard Disk
            Capacity: 4645.6 MB = 4.5 GB (9514260 x 512)
  Device 1: not available
  Device 2: Vendor: CREATIVE Prod.: CD5233E Rev: 2.02
            Type: Removable CD ROM
            Capacity: 450.9 MB = 0.4 GB (230864 x 2048)
  Device 3: not available
SCSI: scanning bus for devices...
  Device 0: (3:0) Vendor: MPL Prod.: MC-DISK-E[0] Rev: 3.9g
            Type: Removable Hard Disk
            Capacity: 496.1 MB = 0.5 GB (1016063 x 512)
  Device 1: (3:3) Vendor: MPL Prod.: MC-DISK-E[1] Rev: 3.9g
            Type: Removable Hard Disk
            Capacity: not available
DOC:   No DiskOnChip found
Rev:   PIP405-1 Rev A PLD0 0 PLD1 0
In:    serial
Out:   serial
Err:   serial
=>
```

3.1.1 Information of the start-up screen

The information on the start-up screen is easily to identify:

- U-Boot version number and build date
- CPU type and strapping settings
- Board type and serial number
- DRAM Size
- Flash Size and Boot source
- IDE devices
- SCSI devices
- Disk On Chip Devices
- Additional Information
- Console assignment

3.2 U-Boot Command Interface

3.2.1 Help system

If you type ***help*** on the U-Boot prompt, you get a list with all possible commands for your board. Please note that the commands may be abbreviated, as long as the command can be clearly identified. Since the development of the U-Boot is still going on, following list represents only the version 0.3.2 for the PIP405 of the U-Boot:

```
=> help
autoscr - run script from memory
base    - print or set address offset
bdinfo  - print Board Info structure
bootelf - Boot from an ELF image in memory
bootm   - boot application image from memory
bootp   - boot image via network using BootP/TFTP protocol
bootvx  - Boot vxWorks from an ELF image
bootd   - boot default, i.e., run 'bootcmd'
cmp     - memory compare
coninfo - print console devices and information
cp      - memory copy
crc32   - checksum calculation
date    - get/set/reset date & time
dcache  - enable or disable data cache
dhcp    - invoke DHCP client to obtain IP/boot params
diskboot- boot from IDE device
docboot - boot from DOC device
doc     - Disk-On-Chip sub-system
echo    - echo args to console
eeprom  - EEPROM sub-system
erase   - erase FLASH memory
fdcboot - boot from floppy device
flinfo  - print FLASH memory information
getdcr  - Get an IBM PPC 4xx DCR's value
go      - start application at address 'addr'
help    - print online help
icache  - enable or disable instruction cache
imd     - i2c memory display
imm     - i2c memory modify (auto-incrementing)
inm     - memory modify (constant address)
imw     - memory write (fill)
in      - read data from an IO port
icrc32  - checksum calculation
iprobe  - probe to discover valid I2C chip addresses
iloop   - infinite loop on address range
isdram  - print SDRAM configuration information
ide     - IDE sub-system
iminfo  - print header information for application image
irqinfo - print information about IRQs
loadb   - load binary file over serial line (kermit mode)
loads   - load S-Record file over serial line
loop    - infinite loop on address range
md      - memory display
mii     - MII utility commands
mm      - memory modify (auto-incrementing)
mtest   - simple RAM test
mw      - memory write (fill)
nm      - memory modify (constant address)
out     - write datum to IO port
pci     - list and access PCI Configuraton Space
printenv- print environment variables
protect - enable or disable FLASH write protection
rarpboot- boot image via network using RARP/TFTP protocol
reginfo - print register information
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
saveenv - save environment variables to persistent storage
saves   - save S-Record file over serial line
scsiboot- boot from SCSI device
scsi    - SCSI sub-system
setdcr  - Set an IBM PPC 4xx DCR's value
setenv  - set environment variables
```

© 2003 by MPL AG

```
sleep    - delay execution for some time
tftpboot- boot image via network using TFTP protocol
               and env variables ipaddr and serverip
usbboot - boot from USB device
usb      - USB sub-system
version - print monitor version
pip405   - PIP405 specific Cmds
?        - alias for 'help'
```

Please note that the command `pip405` is the “Board specific Command”, which is the board name.

To get help on a specific command type ***`help <command>`***. Example:

```
=> help ide
ide reset - reset IDE controller
ide info  - show available IDE devices
ide device [dev] - show or set current device
ide part [dev] - print partition table of one or all IDE devices
ide read  addr blk# cnt
ide write addr blk# cnt - read/write `cnt' blocks starting at block `blk#'
    to/from memory address `addr'

=>
```

3.2.2 Command modifiers

Some of the commands uses modifier after the command, separated with a ‘.’ . These modifier specify the base of the count or size parameter. Following modifier exists:

`.b` Byte (8Bit)
`.w` Word (16Bit)
`.l` Long (32Bit) **Default**

Examples:
Display 16 longs from address 0xFFC00000

```
=> md.l ffc00000 10
ffc00000: 27051956 65d6b450 3cbc2652 003738ef    '..Ve..P<.&R.78.
ffc00010: 00000000 00000000 43450692 05070401    ........CE......
ffc00020: 6b65726e 656c2061 6e642069 6e697472    kernel and initr
ffc00030: 6420666f 72205049 50343035 00000000    d for PIP405....
=>
```

This is the default base, you can also issue the command without `.l`

Display 16 words from address 0xFFC00000

```
=> md.w ffc00000 10
ffc00000: 2705 1956 65d6 b450 3cbc 2652 0037 38ef    '..Ve..P<.&R.78.
ffc00010: 0000 0000 0000 0000 4345 0692 0507 0401    ........CE......
=>
```

Display 16 bytes from address 0xFFC00000

```
=> md.b ffc00000 10
ffc00000: 27 05 19 56 65 d6 b4 50 3c bc 26 52 00 37 38 ef    '..Ve..P<.&R.78.
=>
```

Please note that also the displayed format change according to the base modifier.

© 2003 by MPL AG

3.3 U-Boot Commands

Following chapters describes the most used commands in alphabetic order.

3.3.1 base - print or set address offset

Abbreviation: ***ba***

```
=> help base
base
    - print address offset for memory commands
base off
    - set address offset for memory commands to 'off'

=>
```

With this command you can set the base address of the memory region you like to access. When set, you need only to work with the offsets to this address. Example:

```
=> ba
Base Address: 0x00000000
=> md ffc00000 20
ffc00000: 27051956 65d6b450 3cbc2652 003738ef    '..Ve..P<.&R.78.
ffc00010: 00000000 00000000 43450692 05070401    ........CE......
ffc00020: 6b65726e 656c2061 6e642069 6e697472    kernel and initr
ffc00030: 6420666f 72205049 50343035 00000000    d for PIP405....
ffc00040: 000e3231 002906af 00000000 1f8b0808    ..21.)..........
ffc00050: beb7ba3c 0203766d 6c696e75 7800e45b    ...<..vmlinux..[
ffc00060: 0d745465 7a7eefcc 042624ca 20131925    .tTez~...&$. ..%
ffc00070: 2413889a 90498c1a b6779249 98846ccf    $....I...w.I..l.
=> ba ffc00000
Base Address: 0xffc00000
=> md 0 20
ffc00000: 27051956 65d6b450 3cbc2652 003738ef    '..Ve..P<.&R.78.
ffc00010: 00000000 00000000 43450692 05070401    ........CE......
ffc00020: 6b65726e 656c2061 6e642069 6e697472    kernel and initr
ffc00030: 6420666f 72205049 50343035 00000000    d for PIP405....
ffc00040: 000e3231 002906af 00000000 1f8b0808    ..21.)..........
ffc00050: beb7ba3c 0203766d 6c696e75 7800e45b    ...<..vmlinux..[
ffc00060: 0d745465 7a7eefcc 042624ca 20131925    .tTez~...&$. ..%
ffc00070: 2413889a 90498c1a b6779249 98846ccf    $....I...w.I..l.
=>
```

3.3.2 bdinfo - print Board Info structure

Shows the Board Information Structure which is used to pass information from the U-Boot to the Operating System.
Abbreviation: ***bdi***

```
=>bdi
memstart    = 0x00000000
memsize     = 0x04000000
flashstart  = 0xFFF80000
flashsize   = 0x00400000
flashoffset = 0x00048B00
sramstart   = 0x00000000
sramsize    = 0x00000000
bootflags   = 0xF0000000
procfreq    =    198 MHz
plb_busfreq =     99 MHz
pci_busfreq =     33 MHz
ethaddr     = 00:60:C2:08:00:00
IP addr     = 192.168.3.31
baudrate    =   9600 bps
=>
```

© 2003 by MPL AG

3.3.3 bootd - boot default, i.e run 'bootcmd'

Abbreviation: ***bootd***

This command is an alias to the command 'run bootcmd'. If invoked, the commands in the environment variable `bootcmd` will be executed. This variable contains the necessary commands to boot the default way. Example:

```
=> printenv bootcmd
bootcmd=usb stop; diskboot 400000 0:1; bootm
=>
```

run bootd:

```
=> bootd
stopping USB..

Loading from IDE device 0, partition 1: Name: hda1
  Type: U-Boot
   Image Name:   Linux- mip405
   Created:      2003-05-20  20:56:01 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    698265 Bytes = 681.9 kB
   Load Address: 00000000
   Entry Point:  00000000
## Booting image at 00400000 ...
   Image Name:   Linux- mip405
   Created:      2003-05-20  20:56:01 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    698265 Bytes = 681.9 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
Linux version 2.4.21-rc2 (root@localhost) (gcc version 3.1 20020604 (Red Hat Linux
 Rawhide 3.1-6)) #1 Tue May 20 18:59:04 UTC 2003
```

3.3.4 bootm - boot application image from memory

This command is used to execute the u-boot image residing in the Memory.

Abbreviation: ***bootm***

```
=> help bootm
bootm [addr [arg ...]]
    - boot application image stored in memory
        passing arguments 'arg ...'; when booting a Linux kernel,
        'arg' can be the address of an initrd image

=>
```

Information about the type of operating system, compression, load and entry point are retrieved from the image header. If loading a Linux kernel and a RAMdisk the address of the RAMdisk image is the second argument.

Example:

```
=> bootm 400000
## Booting image at 00400000 ...
   Image Name:   Linux- mip405
   Created:      2003-05-20  20:56:01 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    698265 Bytes = 681.9 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
Linux version 2.4.21-rc2 (root@localhost) (gcc version 3.1 20020604 (Red Hat Linux
 Rawhide 3.1-6)) #1 Tue May 20 18:59:04 UTC 2003
On node 0 totalpages: 32768
zone(0): 4096 pages.
zone(1): 28672 pages.
zone(2): 0 pages.
Kernel command line: console=ttyS0,9600 root=/dev/hda5
Calibrating delay loop... 395.67 BogoMIPS
```

3.3.5 bootp - boot image via network using BOOTP/TFTP protocol

Used to load a image via network using the BOOTP/TFTP protocol.
Abbreviation: ***bootp***

```
=> help bootp
bootp [loadAddress] [bootfilename]
=>
```

To use this command, you need to set-up a bootp server.

3.3.6 cmp - memory compare

The cmp command compares the first memory area with the second one. The command stops on the first non matching address or if the count parameter is reached.
Abbreviation: ***cmp***

```
=> help cmp
cmp [.b, .w, .l] addr1 addr2 count
    - compare memory
=>
```

Example:

```
=> cmp ffc00000 400000 400
word at 0xffc00320 (0x13f0efff) != word at 0x00400320 (0x55ff55ff)
Total of 200 words were the same
=>
```

3.3.7 coninfo - print console devices and information

Abbreviation: ***conin***

```
=>conin
List of available devices:
vga       80000002 S.O
serial    80000003 SIO stdin stdout stderr
=>
```

This command shows information of the console devices. The characters ‘SIO’ shows the device capabilities:
S = System device, I = Input and O = Output. The current assignment is shown by ‘stdin’, ‘stdout’ and ‘stderr’.

3.3.8 cp - memory copy

The copy command copies data from one memory address to another. It distinguish itself if the destination lies in the flash area, so this command can be used to write to the flash.
Abbreviation: ***cp***

```
=> help cp
cp [.b, .w, .l] source target count
    - copy memory
```

Example:
Copy 0x400 Longs from 0xFFC00000 to 0x400000.

```
=> cp ffc00000 400000 400
=>
```

Writing to the flash:

```
=> cp.b 400000 ffc00000 10000
Copy to Flash... done
=>
```

Please note, that the count value specifies the amount of data in the modifier size. This means, count is the number of bytes to copy if the modifier is ‘.b’, the number of words (2Bytes) if the modifier is ‘.w’ and the number of longs (4Bytes) if no modifier or the modifier is ‘.l’.

Note: Please make sure to use the correct modifier (`.b`, `.w` or `.l`). If you are using the wrong modifier, you may overwrite a memory area which is already used.

The flash area must be erased and unprotected, prior to write.

3.3.9 crc32 - checksum calculation

The crc32 command is used to calculate the crc32 checksum over a certain memory area.
Abbreviation: ***crc***

```
=> help crc32
crc32 address count [addr]
    - compute CRC32 checksum [save at addr]
```

Example:

```
=> crc ffc00000 400
CRC32 for ffc00000 ... ffc003ff ==> 056ec8f3
```

If issued with 3 parameters, the crc32 checksum is written to the specified location.

```
=> crc ffc00000 400 400000
CRC32 for ffc00000 ... ffc003ff ==> 056ec8f3
=> md 400000 1
00400000: 056ec8f3    .n..
=>
```

3.3.10 date - get/set/reset date & time

The date command gets, sets or resets date and time of the RTC:
Abbreviation: ***dat***

```
=> help dat
date [MMDDhhmm[[CC]YY][.ss]]
date reset
  - without arguments: print date & time
  - with numeric argument: set the system date & time
  - with 'reset' argument: reset the RTC

=>
```

The command

```
=> date 062913302001
```

sets the RTC to the Jun 29 2001 - 13:30:00

Caution: If the battery has been replaced or the RTC is stuck of another reason, you must first reset the RTC with the command `date reset`.

3.3.11 dhcp - invoke DHCP client to obtain IP/boot params

Used to get the IP and boot parameter using a DHCP server.
Abbreviation: ***dhcp***

```
=> help dhcp
dhcp
=>
```

3.3.12 diskboot- boot from IDE device

Used to load an image from the IDE disk. The U-Boot reads the image by first reads the first block of the partition and then analyzes the image header to distinguish how many blocks must be read. Then the whole image will be read. This means that one or more small partitions must be created on the IDE disk containing only the raw image (no filesystem).
Please note that the device number starts with 0 whereas the partition number starts with 1.
Abbreviation: ***disk***

```
=> help diskboot
diskboot loadAddr dev:part
=>
```

Example:

```
=> diskboot 400000 0:1

Loading from IDE device 0, partition 1: Name: hda1
  Type: U-Boot
   Image Name:   Linux-2.4.2-ppc405gp for PIP405
   Created:      2001-06-08  10:57:25 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    688383 Bytes = 672 kB = 0 MB
   Load Address: 00000000
   Entry Point:  00000000
=>
```

Booting from a CD-ROM is slightly different, since CD-ROMs doesn't have partitions. Only CD-ROMs with bootfile (no harddisk or floppy emulation) are supported.

```
=> ide part
Partition Map for IDE device 1  --   Partition Type: ISO

Part    Start     Sect x Size Type
  0   194229     1744   2048 U-Boot
  1   190743     1741   2048 U-Boot
=>
```

The term 'Part' denotes the image number.
Please note that the image number starts with 0.

```
=> diskboot 400000 1:0

Loading from IDE device 1, partition 0: Name: (C) 2002 MPL AG Switzer  Type: U-Boot
   Image Name:   kernel and initrd for PIP405
   Created:      2002-04-15  11:29:27 UTC
   Image Type:   PowerPC Linux Multi-File Image (gzip compressed)
   Data Size:    3571541 Bytes = 3487 kB = 3 MB
   Load Address: 00000000
   Entry Point:  00000000
   Contents:
   Image 0:   930353 Bytes = 908 kB = 0 MB
   Image 1:  2641173 Bytes = 2579 kB = 2 MB
=>
```

3.3.13 docboot - boot from DOC device

The DiskOnChip boot uses similar parameters as the diskboot.
Abbreviation: ***docb***

```
=> help docb
docboot loadAddr dev
=>
```

3.3.14 doc - Disk-On-Chip sub-system

The doc commands are used for manipulating DiskOnChip devices.
Abbreviation: ***doc***

```
=> help doc
doc info  - show available DOC devices
doc device [dev] - show or set current device
doc read  addr off size
doc write addr off size - read/write `size' bytes starting at offset `off'
    to/from memory address `addr'
doc erase off size - erase `size' bytes of DOC from offset `off'

=>
```

3.3.15 erase - erase FLASH memory

The erase command is used to erase the flash before writing to it.
Abbreviation: ***era***

```
=> help erase
erase start end
    - erase FLASH from addr 'start' to addr 'end'
erase N:SF[-SL]
    - erase sectors SF-SL in FLASH bank # N
erase bank N
    - erase FLASH bank # N
erase all
    - erase all FLASH banks

=>
```

Example:

```
=> era FFC00000 FFF7FFFF
Erase Flash from 0xffc00000 to 0xfff7ffff
..................................................... done
Erased 56 sectors
=>
```

Please note that the 'start' and 'end' parameter must contain the exactly start and endaddress of the sector.

© 2003 by MPL AG

You can also use sector address to erase a portion of the flash. The flash banks are numbered starting with 1, while flash sectors start with 0.

```
=> era 1:0-55
Erase Flash Sectors 0-55 in Bank # 1
..................................................... done
=>
```

3.3.16 fdcboot - boot from floppy device

The floppy disk boot uses similar parameters as the diskboot. Please note that the image on the floppy is also raw, the floppy doesn't contain a file system. If you use linux, you can invoke following command to create such a raw floppy:

```
bash # dd if=IMAGE_FILE_NAME of=/dev/fd0
```

Whereas IMAGE_FILE_NAME is the filename of the image.
Abbreviation: ***fdcb***

```
=> help fdcb
fdcboot loadAddr drive

=>
```

3.3.17 flinfo - print FLASH memory information

Abbreviation: ***fli***

```
=>fli
Bank # 1: Intel TE28F320C3 (32 Mbit, top sector size)
  Size: 4096 KB in 71 Sectors
  Sector Start Addresses:
    FFC00000 E    FFC10000 E    FFC20000 E    FFC30000 E    FFC40000 E
    FFC50000 E    FFC60000 E    FFC70000 E    FFC80000 E    FFC90000 E
    FFCA0000 E    FFCB0000 E    FFCC0000 E    FFCD0000 E    FFCE0000 E
    FFCF0000 E    FFD00000 E    FFD10000 E    FFD20000 E    FFD30000 E
    FFD40000 E    FFD50000 E    FFD60000 E    FFD70000 E    FFD80000 E
    FFD90000 E    FFDA0000 E    FFDB0000 E    FFDC0000 E    FFDD0000 E
    FFDE0000 E    FFDF0000 E    FFE00000 E    FFE10000 E    FFE20000 E
    FFE30000 E    FFE40000 E    FFE50000 E    FFE60000 E    FFE70000 E
    FFE80000 E    FFE90000 E    FFEA0000 E    FFEB0000 E    FFEC0000 E
    FFED0000 E    FFEE0000 E    FFEF0000 E    FFF00000 E    FFF10000 E
    FFF20000 E    FFF30000 E    FFF40000 E    FFF50000 E    FFF60000 E
    FFF70000 E    FFF80000  RO  FFF90000  RO  FFFA0000  RO  FFFB0000  RO
    FFFC0000 ERO  FFFD0000 ERO  FFFE0000 ERO  FFFF0000 ERO  FFFF2000 ERO
    FFFF4000 ERO  FFFF6000 ERO  FFFF8000 ERO  FFFFA000 ERO  FFFFC000 ERO
    FFFFE000  RO
=>
```

This command shows the start address of each flash sector. The character 'E' indicates that the sector is erased, and the characters 'RO' indicates that this sector is read only.

3.3.18 fsinfo - print information about filesystems

This command is used to get information about the filesystem residing in the NAND flash. To install a filesystem in the NAND flash, you have to make a JFFS2 image and then write this image to the NAND flash with the command `nand write.jffs2` please refer to chapter 3.3.29 nand - NAND sub-system. For more information see also on www.mpl.ch.
Abbreviation: ***fsi***

```
=> help fsi
fsinfo     - print information about filesystems

=>
```

3.3.19 fsload - load binary file from a filesystem image

Used for NAND flash filesystem.
Abbreviation: ***fsl***

```
=> help fsl
fsload [ off ] [ filename ]
    - load binary file from flash bank
      with offset 'off'
=>
```

3.3.20 ide - IDE sub-system

The IDE subsystem is controlled with the ide command.
Abbreviation: ***ide***

```
=> help ide
ide reset - reset IDE controller
ide info  - show available IDE devices
ide device [dev] - show or set current device
ide part [dev] - print partition table of one or all IDE devices
ide read  addr blk# cnt
ide write addr blk# cnt - read/write `cnt' blocks starting at block `blk#'
    to/from memory address `addr'
=>
```

ide reset reinitializes the IDE controller and shows the available devices:

```
=> ide reset

Reset IDE: Bus 0: OK Bus 1: not available
  Device 0: Model: TOSHIBA MK2104MAV Firm: E4.02 A Ser#: 68I32705
            Type: Hard Disk
            Capacity: 2067.1 MB = 2.0 GB (4233600 x 512)
  Device 1: not available
=>
```

ide info shows the available devices
ide device shows/sets the current device. This is used with the commands ide read, ide write end ide part.

```
=> ide dev

IDE device 0: Model: TOSHIBA MK2104MAV Firm: E4.02 A Ser#: 68I32705
            Type: Hard Disk
            Capacity: 2067.1 MB = 2.0 GB (4233600 x 512)
=>
```

ide part shows the partition table of the current device or of the device specified by the parameter:

```
=> ide part 0

Partition Map for IDE device 0  --   Partition Type: DOS

Partition     Start Sector     Num Sectors     Type
    1                63           19089       4
    2             19152           19152       4
    3             38304         4195296       5 Extd
    5             38367         4094433      83
    6           4132863          100737      82
=>
```

ide read/write are used to read from / write to the current device. These commands are rarely used. To boot from ide use the command 3.3.12 diskboot- boot from IDE device instead.

Note for CD-ROMs:
CD-ROMs (ATAPI Devices) don't contain a partition table, but they may contain one or more bootrecords. Supported are only boot images with no emulation. The bootrecords are listed like partitions on a harddisk but with the first boot entry number starting at 0.

3.3.21 isdram - print SDRAM configuration information

SDRAM configuration information is only available if the SDRAM consist of a DIMM. It prints the values from the SPD EEPROM of the SDRAM DIMM.
Abbreviation: ***isdram***

```
=> help isdram
isdram chip
    - print SDRAM configuration information
(valid chip values 50..57)
=>
```

An example of this command is not provided, because this command is seldom used and the information relays only on the DIMM manufacturer.

© 2003 by MPL AG

3.3.22 iminfo - print header information for application image

Print information about the loaded image in the memory. The images are created with the `mkimage` command, see Chapter 3.5.4 Creating bootable kernel images.
Abbreviation: ***imi***

```
=> help imi
iminfo addr [addr ...]
    - print header information for application image starting at
      address 'addr' in memory; this includes verification of the
image contents (magic number, header and payload checksums)
```

Example, information about a kernel image residing in the flash of a PIP405.

```
=> imi ffc00000
## Checking Image at ffc00000 ...
   Image Name:   kernel and initrd for PIP405
   Created:      2002-04-16  13:25:38 UTC
   Image Type:   PowerPC Linux Multi-File Image (gzip compressed)
   Data Size:    3619055 Bytes =  3.5 MB
   Load Address: 00000000
   Entry Point:  00000000
   Contents:
   Image 0:   930353 Bytes = 908.5 kB
   Image 1:  2688687 Bytes =  2.6 MB
   Verifying Checksum ... OK
=>
```

3.3.23 irqinfo - print information about IRQs

Abbreviation: ***irq***

```
=> irq

Interrupt-Information:
Nr  Routine   Arg       Count
25  03fca2d0  00000000  5
31  03fcbcb8  00000000  59528
=>
```

The Nr indicates the Interrupt line, routine shows the address of the interrupt handler, arg shows with which parameters the routine is called an the count shows the number of handled interrupt.
Please note that the U-Boot is designed to access all devices without interrupt whenever possible. So for most configurations the list is empty.

3.3.24 loadb - load binary file over serial line (kermit mode)

Used to receive the image via the serial line in kermit mode.
Abbreviation: ***loadb***

```
=> help loadb
loadb [ off ] [ baud ]
    - load binary file over serial line with offset 'off' and baudrate 'baud'
=>
```

Example:

```
=> loadb 400000 57600
## Switch baudrate to 57600 bps and press ENTER ...
```

Here you have to change the baudrate of your terminal to 57600. Then press enter.

```
## Ready for binary (kermit) download to 0x00400000 at 57600 bps...
```

Now you can send the image in kermit mode, please refer to the documentation of your terminal program how to do this.

```
## Total Size      = 0x000187e6 = 100326 Bytes
## Start Addr      = 0x00400000
## Switch baudrate to 9600 bps and press ESC ...
```

Here you have to change the baudrate of your terminal back to 9600. Then press escape.

```
=>
```

3.3.25 md - memory display

Displays the memory contents on the given address.
Abbreviation: ***md***

```
=> help md
md [.b, .w, .l] address [# of objects]
    - memory display
=>
```

Example:
Display 16 longs from address 0xFFC00000

```
=> md.l ffc00000 10
ffc00000: 27051956 65d6b450 3cbc2652 003738ef    '..Ve..P<.&R.78.
ffc00010: 00000000 00000000 43450692 05070401    ........CE......
ffc00020: 6b65726e 656c2061 6e642069 6e697472    kernel and initr
ffc00030: 6420666f 72205049 50343035 00000000    d for PIP405....
=>
```

Display 16 words from address 0xFFC00000

```
=> md.w ffc00000 10
ffc00000: 2705 1956 65d6 b450 3cbc 2652 0037 38ef    '..Ve..P<.&R.78.
ffc00010: 0000 0000 0000 0000 4345 0692 0507 0401    ........CE......
=>
```

Display 16 bytes from address 0xFFC00000

```
=> md.b ffc00000 10
ffc00000: 27 05 19 56 65 d6 b4 50 3c bc 26 52 00 37 38 ef    '..Ve..P<.&R.78.
=>
```

3.3.26 mm - memory modify (auto-incrementing)

With the memory modify command you can modify a memory area. If you press RETURN when the command prompts, the memory contents is unchanged. The command stops as soon you enter an other number than hexadecimal, like '.'.
Abbreviation: ***mm***

```
=> help mm
mm [.b, .w, .l] address
    - memory modify, auto increment address
=>
```

Example:

```
=> md.b 400000
00400000: 55 aa 30 31 65 d6 b4 50 3c bc 26 52 00 37 38 ef    U.01e..P<.&R.78.
=> mm.b 400000
00400000: 55 ?
00400001: aa ? 55
00400002: 30 ? 31
00400003: 31 ? 32
00400004: 65 ? 33
00400005: d6 ? 34
00400006: b4 ? 35
00400007: 50 ? 36
00400008: 3c ?
00400009: bc ? 37
0040000a: 26 ? .
=> md.b 400000
00400000: 55 55 31 32 33 34 35 36 3c 37 26 52 00 37 38 ef    UU123456<7&R.78.
=>
```

3.3.27 mtest - simple RAM test

This command executes a RAM test.
Abbreviation: ***mt***

```
=> help mtest
mtest [start [end [pattern]]]
    - simple RAM read/write test
```

Example:

```
=> mtest 100000 400000
Testing 00100000 ... 00400000:
Pattern 0000000F  Writing...  Reading...
=>
```

Note: **This test uses a destructive testing, which means that the tested memory contents will be destroyed. Be careful not to overwrite used memory.**

© 2003 by MPL AG

3.3.28 mw - memory write (fill)

The memory write command is used to fill a memory area with a pattern. Depending on the modifier, the pattern size is a byte (8 Bit), word (16 Bit) or a long (32 Bit). Note that the count parameter has also the base of the modifier.
Abbreviation: ***mw***

```
=> help mw
mw [.b, .w, .l] address value [count]
    - write memory
=>
```

Example:

```
=> md.b 400000 10
00400000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
=> mw.b 400000 12 10
=> md.b 400000 10
00400000: 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12    ................
=> mw.w 400000 3456 8
=> md.w 400000 8
00400000: 3456 3456 3456 3456 3456 3456 3456 3456    4V4V4V4V4V4V4V4V
=> mw.l 400000 7890abcd 4
=> md.l 400000 4
00400000: 7890abcd 7890abcd 7890abcd 7890abcd    x...x...x...x...
=>
```

3.3.29 nand - NAND sub-system

The NAND sub-system is used to handle with the NAND flash.
Abbreviation: ***na***

```
=> help na
nand info  - show available NAND devices
nand device [dev] - show or set current device
nand read[.jffs2]  addr off size
nand write[.jffs2] addr off size - read/write `size' bytes starting
    at offset `off' to/from memory address `addr'
nand erase [clean] [off size] - erase `size' bytes from
    offset `off' (entire device if not specified)
nand bad - show bad blocks
nand read.oob addr off size - read out-of-band data
nand write.oob addr off size - read out-of-band data
=>
```

`nand info` shows the information of the connected NAND flash:

```
=> nand info
Device 0: Samsung KM29U64000 (ECE6) at 0x4e000000 ( 8 MB,  8 kB sector)
=>
```

`nand erase` erases a part or the entire NAND flash:

```
=> nand erase 0 100000
NAND erase: device 0 offset 0, size 1048576 ... OK
=>
```

`nand write.jffs2` writes the specified memory contents to the NAND flash.
The `.jffs2` modifier makes sure that the bad blocks are not used. If you use this command without the `.jffs2` modifier, portions of the image may be lost (written in bad blocks).

```
=> nand write.jffs2 33000000 0 $filesize
NAND write: device 0 offset 0, size 778465 ... 778465 bytes write: OK
=>
```

Images written with this command are used in conjunction with the `nboot` command, see following chapter.

3.3.30 nboot - boot from NAND device

Used to load the image residing in the NAND flash. To write an image into the NAND flash, refer to the previous chapter.
Abbreviation: ***nb***

```
=> help nboot
nboot loadAddr dev
=>
```

Example:

```
=> nboot 33000000 0
Loading from device 0: <NULL> at 0x4E000000 (offset 0x0)
Image Name: Linux-2.4.19-rmk4-swl6-dm1
Created: 2003-04-17 14:49:55 UTC
Image Type: ARM Linux Kernel Image (gzip compressed)
Data Size: 778401 Bytes = 760.2 kB
Load Address: 30008000
Entry Point: 30008000
=>
```

3.3.31 nm - memory modify (constant address)

With the nm command you can write to a constant memory location. It works like the mm command with the difference, that the address is not incremented.
Abbreviation: ***nm***

```
=> help nm
nm [.b, .w, .l] address
    - memory modify, read and keep address
=>
```

Example:

```
=> nm.b 400000
00400000: 00 ? 80
00400000: 80 ? 81
00400000: 81 ? 01
00400000: 01 ? 40
00400000: 40 ? .
=> md.b 400000 1
00400000: 40    @
=>
```

3.3.32 pci - list and access PCI Configuration Space

Abbreviation: ***pci***

```
=> help pci
pci [bus] [long]
    - short or long list of PCI devices on bus 'bus'
pci header b.d.f
    - show header of PCI device 'bus.device.function'
pci display[.b, .w, .l] b.d.f [address] [# of objects]
    - display PCI configuration space (CFG)
pci next[.b, .w, .l] b.d.f address
    - modify, read and keep CFG address
pci modify[.b, .w, .l] b.d.f address
    -  modify, auto increment CFG address
pci write[.b, .w, .l] b.d.f address value
    - write to CFG address
```

Example for a PIP405:

```
=> pci
Scanning PCI devices on bus 0
BusDevFun  VendorId   DeviceId   Device Class       Sub-Class
_____________________________________________________________
00.00.00   0x1014     0x0156     Bridge device           0x00
00.06.00   0x8086     0x7110     Bridge device           0x01
00.06.01   0x8086     0x7111     Mass storage controller 0x01
00.06.02   0x8086     0x7112     Serial bus controller   0x03
00.06.03   0x8086     0x7113     Bridge device           0x80
00.07.00   0x102c     0x00c0     Display controller      0x00
00.08.00   0x1000     0x0001     Mass storage controller 0x00
```

3.3.33 printenv- print environment variables

Abbreviation: ***pri***

```
=> help pri
printenv
    - print values of all environment variables
printenv name ...
    - print value of environment variable 'name'
=>
```

Example:

```
=> printenv
bootargs=console=ttyS0,9600 root=/dev/hda5
baudrate=9600
loads_echo=1
preboot=
serial#=MIP405_100
ethaddr=00.60.C2.0A.00.5E
bootdelay=-1
ipaddr=192.168.3.37
serverip=192.168.3.17
bootyard=setenv bootargs console=ttyS0,9600 root=/dev/ram0 ide=nodma; bootm ffc00000
bootcmd=usb stop; diskboot 400000 0:1; bootm

Environment size: 348/2044 bytes
=>
```

3.3.34 protect - enable or disable FLASH write protection

The protect command enables or disables the protection of the specified sectors

Abbreviation: ***prot***

```
=> help prot
protect on  start end
    - protect FLASH from addr 'start' to addr 'end'
protect on  N:SF[-SL]
    - protect sectors SF-SL in FLASH bank # N
protect on  bank N
    - protect FLASH bank # N
protect on  all
    - protect all FLASH banks
protect off start end
    - make FLASH from addr 'start' to addr 'end' writable
protect off N:SF[-SL]
    - make sectors SF-SL writable in FLASH bank # N
protect off bank N
    - make FLASH bank # N writable
protect off all
    - make all FLASH banks writable
```

Example:

```
=> fli
Bank # 1: Intel TE28F320C3 (32 Mbit, top sector size)
  Size: 4096 KB in 71 Sectors
  Sector Start Addresses:
    FFC00000      FFC10000      FFC20000      FFC30000      FFC40000
    FFC50000      FFC60000      FFC70000      FFC80000      FFC90000
    FFCA0000      FFCB0000      FFCC0000      FFCD0000      FFCE0000
    FFCF0000      FFD00000      FFD10000      FFD20000      FFD30000
    FFD40000      FFD50000      FFD60000      FFD70000      FFD80000
    FFD90000      FFDA0000      FFDB0000      FFDC0000      FFDD0000
    FFDE0000      FFDF0000      FFE00000      FFE10000      FFE20000
    FFE30000      FFE40000      FFE50000      FFE60000      FFE70000
    FFE80000      FFE90000      FFEA0000      FFEB0000      FFEC0000
    FFED0000      FFEE0000      FFEF0000      FFF00000      FFF10000
    FFF20000      FFF30000      FFF40000      FFF50000      FFF60000
    FFF70000      FFF80000  RO  FFF90000  RO  FFFA0000  RO  FFFB0000  RO
    FFFC0000  RO  FFFD0000 E    FFFE0000 E    FFFF0000 E    FFFF2000 E
    FFFF4000 E    FFFF6000 E    FFFF8000 E    FFFFA000 E    FFFFC000 E
    FFFFE000
=> prot on all
Protect Flash Bank # 1
=> fli
Bank # 1: Intel TE28F320C3 (32 Mbit, top sector size)
  Size: 4096 KB in 71 Sectors
  Sector Start Addresses:
    FFC00000  RO  FFC10000  RO  FFC20000  RO  FFC30000  RO  FFC40000  RO
```

```
    FFC50000  RO  FFC60000  RO  FFC70000  RO  FFC80000  RO  FFC90000  RO
    FFCA0000  RO  FFCB0000  RO  FFCC0000  RO  FFCD0000  RO  FFCE0000  RO
    FFCF0000  RO  FFD00000  RO  FFD10000  RO  FFD20000  RO  FFD30000  RO
    FFD40000  RO  FFD50000  RO  FFD60000  RO  FFD70000  RO  FFD80000  RO
    FFD90000  RO  FFDA0000  RO  FFDB0000  RO  FFDC0000  RO  FFDD0000  RO
    FFDE0000  RO  FFDF0000  RO  FFE00000  RO  FFE10000  RO  FFE20000  RO
    FFE30000  RO  FFE40000  RO  FFE50000  RO  FFE60000  RO  FFE70000  RO
    FFE80000  RO  FFE90000  RO  FFEA0000  RO  FFEB0000  RO  FFEC0000  RO
    FFED0000  RO  FFEE0000  RO  FFEF0000  RO  FFF00000  RO  FFF10000  RO
    FFF20000  RO  FFF30000  RO  FFF40000  RO  FFF50000  RO  FFF60000  RO
    FFF70000  RO  FFF80000  RO  FFF90000  RO  FFFA0000  RO  FFFB0000  RO
    FFFC0000  RO  FFFD0000 ERO  FFFE0000 ERO  FFFF0000 ERO  FFFF2000 ERO
    FFFF4000 ERO  FFFF6000 ERO  FFFF8000 ERO  FFFFA000 ERO  FFFFC000 ERO
    FFFFE000  RO
=>
```

Note: **The protection mechanism does not use the hardware protection feature of the flash. It is a software solution, and the protection is volatile, which means that after a reboot only the bootloader is protected.**

3.3.35 rarpboot- boot image via network using RARP/TFTP protocol

Used to load the image via the network using RARP/TFTP protocol.
Abbreviation: ***rarp***

```
=> help rarp
rarpboot [loadAddress] [bootfilename]
=>
```

To use this command, you need to set-up a rarp server.

3.3.36 reset - Perform RESET of the CPU

With the reset command you can restart the board:

```
=> reset
Initializing SDRAM, Please stand by...................


U-Boot 0.4.0 (Jul 11 2003 - 10:33:29)
(c) 2003 by MPL AG Switzerland, MEV-10072-001 released
```

3.3.37 run - run commands in a environment variable

This command is used to execute commands which are stored in a environment variable.

```
=> help run
run var [...]
    - run the commands in the environment variable(s) 'var'

=>
```

Following example shows how to change the boot arguments, and boots an image stored in the flash:

```
=> printenv bootyard
bootyard=setenv bootargs console=ttyS0,9600 root=/dev/ram0 ide=nodma; bootm ffc00000
=> run bootyard
## Booting image at ffc00000 ...
   Image Name:   kernel and initrd for MIP405
   Created:      2002-04-16  13:17:43 UTC
   Image Type:   PowerPC Linux Multi-File Image (gzip compressed)
   Data Size:    3613150 Bytes =  3.4 MB
   Load Address: 00000000
   Entry Point:  00000000
   Contents:
   Verifying Checksum ... OK
   Uncompressing Multi-File Image ... OK
   Loading Ramdisk to 07cd8000, end 07f878ae ... OK
Linux version 2.4.18-rc1-yard-boot (root@m05denis) (gcc version 2.95.3 20010315 (r
elease)) #3 Thu Apr 11 11:23:27 Local time zone must be set--see zic manual page 2
002
On node 0 totalpages: 32768
zone(0): 4096 pages.
zone(1): 28672 pages.
zone(2): 0 pages.
Kernel command line: console=ttyS0,9600 root=/dev/ram0 ide=nodma
ide_setup: ide=nodmaIDE: Prevented DMA
Calibrating delay loop... 395.67 BogoMIPS
```

3.3.38 saveenv - save environment variables to persistent storage

On boot-up the environment is copied from the EEPROM to the RAM, and all changes on the environment variables are done in the RAM. So the environment is volatile. To store the actual environment to the persistent storage use this command.
Abbreviation: ***save***

```
=> save
Saving Environment to EEPROM...
=>
```

3.3.39 scsiboot- boot from SCSI device

Booting from a SCSI device works in the same way as the diskboot command.
Abbreviation: ***scsib***

```
=> help scsib
scsiboot loadAddr dev:part
=>
```

3.3.40 scsi - SCSI sub-system

The SCSI sub-system is similar to the ide sub-system, with the difference that all transactions take place on the SCSI bus and the command starts with scsi.
Abbreviation: ***scsi***

```
=> help scsi
scsi reset - reset SCSI controller
scsi info  - show available SCSI devices
scsi scan  - (re-)scan SCSI bus
scsi device [dev] - show or set current device
scsi part [dev] - print partition table of one or all SCSI devices
scsi read addr blk# cnt - read `cnt' blocks starting at block `blk#'
     to memory address `addr'
=>
```

3.3.41 setenv - set environment variables

Use this command to set or delete an environment variable.
Abbreviation: ***setenv***

```
=> help setenv
setenv name value ...
    - set environment variable 'name' to 'value ...'
setenv name
    - delete environment variable 'name'
```

If you call this command with only one parameter the variable will be deleted:

```
=> printenv bootit
bootit=usb stop; bootd
=> setenv bootit
=> printenv bootit
## Error: "bootit" not defined
=>
```

To set a environment variable, invoke this command with more than one parameter, whereas the first one is the variable name, and the following parameters are the variable contents.
Please note that the variable name must be delimited with a space from the contents.

```
=> printenv bootit
## Error: "bootit" not defined
=> setenv bootit usb stop\; bootd
=> printenv bootit
bootit=usb stop; bootd
=>
```

If assigning more than one command to a variable, use semicolons ';' to delimiting the commands.

Note: To escape special character such as ';' (command delimiter) and '$' (variable substitution) use the backslash '\'.

```
=> setenv update era FFC00000 FFF7FFFF\; tftp 400000 mip405_multi.img\; cp.b 400000 FFC00000
\$(filesize)
=> printenv update
update=era FFC00000 FFF7FFFF; tftp 400000 mip405_multi.img; cp.b 400000 ffc00000 $
(filesize)
=>
```

3.3.42 tftpboot- boot image via network using TFTP protocol

Used to load the image via the network using the TFTP protocol. For this purpose following environment variables must be set:

- `ipaddr` The IP address of the board
- `netmask` The appropriate netmask
- `serverip` The IP address of the TFTP Server

Abbreviation: ***tftp***

```
=> help tftp
tftpboot [loadAddress] [bootfilename]
=>
```

To use this command, you need to set-up a tftp server.
Example:

```
=> tftp 400000 u-boot-MIP405_zip.img
ENET Speed is 100 Mbps - HALF duplex connection
ARP broadcast 1
TFTP from server 192.168.3.17; our IP address is 192.168.3.37
Filename 'u-boot-MIP405_zip.img'.
Load address: 0x400000
Loading: #########################
done
Bytes transferred = 119777 (1d3e1 hex)
=>
```

3.3.43 usb - USB sub-system

The USB system is a little bit more complex than others sub-systems. That is mainly because the USB allows to connect human interface devices as well as storage devices. Along this the USB controller uses buffer from the main memory and updates these buffers all msec. This means that the USB controller arbitrate the PCI bus every msec and reads/writes data from/to the main memory. Starting up a kernel without stooping the USB controller may cause a system crash. So if using the USB with the U-Boot, make sure to stop the USB sub-system before starting the kernel. If you are using an USB keyboard you must create a command in the environment which switches the USB off and then boot the image. See 3.5.3.2 Working with an USB Keyboard for more information.
Abbreviation: ***usb***

```
=> help usb
usb  reset - reset (rescan) USB controller
usb  stop [f]  - stop USB [f]=force stop
usb  tree  - show USB device tree
usb  info [dev] - show available USB devices
usb  scan  - (re-)scan USB bus for storage devices
usb  device [dev] - show or set current USB storage device
usb  part [dev] - print partition table of one or all USB storage devices
usb  read addr blk# cnt - read `cnt' blocks starting at block `blk#'
     to memory address `addr'

=>
```

usb reset or ***usb start*** reinitializes the USB controller and searches all USB devices on the bus:

```
=> usb start
(Re)start USB...
USB:   scanning bus for devices... 5 USB Devices found
=>
```

usb tree shows all the USB devices in a tree structure how the devices are connected:

```
=> usb tree

Device Tree:
  1  Hub (12MBit/s, 0mA)
  |  UHCI Root Hub
  |
  +-2  Mass Storage (12MBit/s, 500mA)
  |    MITSUMI  MITSUMI USB FDD
  |
  +-3  Hub (12MBit/s, 100mA)
    |   Generic USB Hub
    |
    +-4  Human Interface (1.5MBit/s, 100mA)
    |    Chicony KMp
    |
    +-5  Human Interface (1.5MBit/s, 50mA)
         Logitech USB Mouse

=>
```

usb info [dev] shows detailed information about the specified device. if the device is omitted, all the USB devices are showed:

```
=> usb info 4
config for device 4
4: Human Interface,  USB Revision 1.0
 - Chicony KMp
 - Class: (from Interface) Human Interface
 - PacketSize: 8  Configurations: 1
 - Vendor: 0x04f2  Product 0x0001 Version 1.0
   Configuration: 1
   - Interfaces: 2 Bus Powered Remote Wakeup 100mA
     Interface: 0
     - Alternate Settings 0, Endpoints: 1
     - Class Human Interface, Subclass: Boot Keyboard
     - String: "Keyboard"
     - Endpoint 1 In Interrupt MaxPacket 8 Interval 10ms
     Interface: 1
     - Alternate Settings 0, Endpoints: 1
     - Class Human Interface, Subclass: Boot Mouse
     - String: "Mouse"
     - Endpoint 2 In Interrupt MaxPacket 3 Interval 10ms

=>
```

usb scan searches for storage devices on the USB, and creates storage devices:

```
=> usb scan
Scan for storage device:
scanning bus for storage devices...
  Device 0: Vendor: MITSUMI  Prod.: USB FDD          Rev: 1035
            Type: Removable Hard Disk
            Capacity: 1.4 MB = 0.0 GB (2880 x 512)
=>
```

Note: Removable media's must be present for this command.

usb part [dev] lists the partition table of the storage devices

Note: Floppy disk don't have partition tables. They are handled like described in chapter 3.3.16 fdcboot - boot from floppy device.

Supported USB Devices:

- USB Keyboard
- USB Floppy

Other devices may also work, but they are not tested.

© 2003 by MPL AG

3.3.44 usbboot - boot from USB device

Booting from USB is a little bit more complex than from other boot sources. This is because the usb subsystem must first be started. For more information refer to 3.3.43 usb - USB sub-system.

Note: Booting via USB is only tested with floppy, although other devices may also work.

```
=> usb start
(Re)start USB...
USB:   scanning bus for devices... 2 USB Devices found
=>
```

Invoke the command to scan for USB Storage devices:

```
=> usb scan
Scan for storage device:
scanning bus for storage devices...
Device 0: (2:0) Vendor: TEAC     Prod.: FD-05PUB         Rev: 1026
          Type: Removable Hard Disk
          Capacity: 1.4 MB = 0.0 GB (2880 x 512)
```

Now you can use the usb boot command:
Abbreviation: ***usbo***

```
=> help usbbo
usbboot loadAddr dev:part
=>
```

```
=> usbbo 400000 0:0
error reading partinfo...try to boot raw

Loading from USB device 0, partition 0: Name: Raw  Type: U-Boot
First Block: 0,  # of blocks: 2880, Block Size: 512
   Image Name:   Linux-2.4.20
   Created:      2003-04-23  21:42:58 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    724166 Bytes = 707.2 kB
   Load Address: 00000000
   Entry Point:  00000000
.................................................................
=>
```

The ‘error reading partinfo’ can be ignored. since it is possible to boot also from a USB harddisk, the U-Boot tries to read the partition table first.

NOTE: Please make sure that you shut down the usb sub-system before you starting the image in the memory:

```
=> usb stop
```

3.3.45 pip405/mip405/vcma9 - Board specific Cmds

For every MPL SBC a board specific command exists. These are mostly used to program a new U-Boot to the flash, but you can also retrieve information about Board revision, configuration options and PLD revisions.
Example for a VCMA9:

```
=> vcma9 info
Board: VCMA9-1 PCB Rev: A  SN: 102 PLD: 0,0
SDRAM: 4 chips 4Mx8x4
=>
```

Depending on your MPL board different commands are supported. One command which is always available is used to update the u-boot. Please refer to chapter 3.5.1 Updating the bootloader for further information.

3.4 U-Boot Environment Variables

One of the powerful features of the U-Boot are the environment variables. They are stored in the on board EEPROM and protected with a CRC32 checksum. These variables are used to store the board properties and control the boot behavior.

3.4.1 Board properties Variables

Two variables are set during manufacturing, and are neither erasable nor over-write-able:

serial#
contains the Board name and the serial number for example: MIP405_100

ethaddr
contains the MAC address of the ethernet port.

3.4.2 Variables controlling the OS boot

Following variables are used to control the boot process of the operating system:

bootargs
contains the boot arguments which will be transferred to the kernel.

bootcmd
contains the commands with which the operating system is loaded and booted. This variable will be executed if invoking the command ‘bootd’ or the bootdelay as expired.

bootdelay
contains the number seconds u-boot waits, before executing the variable ‘bootcmd’. If set to ‘0’ u-boot starts immediately. If set to a negative number, u-boot does not boot the operating system and prompt the user for input.

3.4.3 Variables for U-Boot behavior

Following variables changes the behavior of the u-boot:

baudrate
contains the baudrate of the serial console. Please note that this number is decimal. If not defined, the baudrate of the serial line is 9600.

loads_echo
Set this variable to ‘1’ if your terminal program requires an echo of the sent data. This is only used in conjunction with the command ‘loadb’.

preboot
contains commands which will be executed before u-boot prompts.

loadaddr
contains the load address where the image from ‘tftpboot’ or ‘loadb’ is stored.

autostart
Set to ‘yes’ if the image should automatically started after it is load.

3.4.4 Network Variables

Following variables are used for network properties:

bootfile
contains the default name of the file which will be retrieved during tftpboot.

ipaddr
contains the IP address of the board.

serverip
contains the IP address of the TFTP server.

autoload
used to decide if the commands ‘dhcp’, ‘rarpb’ or ‘bootp’ should also load the image via tftp. If set to ‘no’ the image will not be loaded.

© 2003 by MPL AG

3.4.5 Automatically updated Variables

filesize

contains the file size in bytes after loading an image into the memory. This can be used when writing an image to the flash.

Depending on your server configuration some of the following variables will be updated after an ‘dhcp’, ‘rarpb’ or ‘bootp’ command:

- **ipaddr**
- **serverip**
- **netmask**
- **gatewayip**
- **dnsip**
- **bootfile**
- **rootpath**
- **hostname**

© 2003 by MPL AG

3.5 Procedures

3.5.1 Updating the bootloader

To update the U-Boot you need a new U-Boot image, which is available on www.mpl.ch. The image may be in a zipped file containing 3 different file format:

- `u-boot-X.Y.Z-BOARD_zip.img`
  This is a compressed bootloader image file.
- `u-boot-X.Y.Z-BOARD.img`
  This is a uncompressed bootloader image file. It is only used for boards with the ppcboot version < 1.1.3
- `u-boot-X.Y.Z-BOARD.bin`
  This is a raw binary file which is only used to reprogram a badly programmed flash.

The U-Boot update is parted in 2 steps:

- load the image in the memory
- program the flash.

Warning: **It may happen, that the environment variables are corrupted after an Update. Therefore it is a good idea to invoke first the command:**

```
=> printenv
```

and write the important variables down. At least the variables `serial#` and `ethaddr` must be rewritten to the environment, if the variables are corrupted after an Update.

3.5.1.1 Load the image in the memory

To load the image in the memory choose your preferred load method (tftp, usb, floppy, serial etc) and use the appropriate load command.
Example with tftpboot:

```
=> tftpboot 400000 u-boot-0.4.0-MIP405_zip.img
ENET Speed is 100 Mbps - HALF duplex connection
ARP broadcast 1
TFTP from server 192.168.3.17; our IP address is 192.168.3.37
Filename 'u-boot-0.4.0-MIP405_zip.img'.
Load address: 0x400000
Loading: #########################
done
Bytes transferred = 119777 (1d3e1 hex)
=>
```

3.5.1.2 Program the flash

To simplify the flash programming procedure a special command is implemented:
`=> [BOARD] flash mem [LOADADDRESS]`
Example on a MIP405:

```
=> mip405 flash mem

updating bootloader image from memory at 400000
   Image Name:   U-Boot-MIP405
   Created:      2003-07-11   9:23:00 UTC
   Image Type:   PowerPC U-Boot Firmware (gzip compressed)
   Data Size:    121288 Bytes = 118.4 kB
   Load Address: fff80000
   Entry Point:  00000000
Verifying Checksum ...   Uncompressing  ...   OK
Erasing at fff80000 (sector 56) (start fff80000)

.............. done
flash erased, programming from 0x500040 0x80000 Bytes
........OK programming done
=>
```

With the reset command you can restart the board:

```
=> reset
Initializing SDRAM, Please stand by..................

U-Boot 0.4.0 (Jul 11 2003 - 10:33:29)
(c) 2003 by MPL AG Switzerland, MEV-10072-001 released
```

3.5.2 Programming a kernel image into the flash

The location whereas the kernel image should written to, is board depending.

MIP405/PIP405:

The flash is used to store the bootloader and the unused flash memory can be used to store a kernel image. The bootloader is programmed from address 0xFFF80000 to 0xFFFFFFFF and must not be overwritten. The remaining flash memory depends on the populated flash. In this manual we assume to have a 4MByte flash, which gives us 3.5MByte free flash memory (from 0xFFC00000 to 0xFFF7FFFF).

VCMA9:

The VCMA9 has a NAND Flash for application data, so the entire NAND flash can be used for Kernel Image.

3.5.2.1 Erase the flash

On a MIP405/PIP405 invoke the command:

```
=> erase FFC00000 FFF7FFFF
Erase Flash from 0xffc00000 to 0xfff7ffff
.......................................................... done
Erased 56 sectors
=>
```

Please note that it is possible that you have first to unprotect the sectors. Refer to 3.3.34 protect - enable or disable FLASH write protection for more information.

On a VCMA9 invoke the command:

```
=> nand erase 0 800000
NAND erase: device 0 offset 0, size 8388608 ... OK
=>
```

3.5.2.2 Load the image into the memory

Load the kernel image with your preferred load method to the memory. Here the example using tftpboot:

```
=> tftpboot 400000 mip405_multi.img
ENET Speed is 100 Mbps - HALF duplex connection
TFTP from server 192.168.3.17; our IP address is 192.168.3.37
Filename 'mip405_multi.img'.
Load address: 0x400000
Loading: ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ############################################################
done
Bytes transferred = 3613214 (37221e hex)
=> imi 400000

## Checking Image at 00400000 ...
   Image Name:   kernel and initrd for MIP405
   Created:      2002-04-16  13:17:43 UTC
   Image Type:   PowerPC Linux Multi-File Image (gzip compressed)
   Data Size:    3613150 Bytes =  3.4 MB
   Load Address: 00000000
   Entry Point:  00000000
   Contents:
   Image 0:   796961 Bytes = 778.3 kB
   Image 1:  2816174 Bytes =  2.7 MB
   Verifying Checksum ... OK
=>
```

3.5.2.3 Write the image into the flash

On MIP405/PIP405 use the command cp to copy the image into the flash. Refer to the chapter 3.3.8 cp - memory copy for more information:

```
=> cp.b 400000 ffc00000 $filesize
Copy to Flash... ...................................................done
=> imi ffc00000

## Checking Image at ffc00000 ...
   Image Name:   kernel and initrd for MIP405
   Created:      2002-04-16  13:17:43 UTC
   Image Type:   PowerPC Linux Multi-File Image (gzip compressed)
   Data Size:    3613150 Bytes =  3.4 MB
   Load Address: 00000000
   Entry Point:  00000000
   Contents:
   Image 0:   796961 Bytes = 778.3 kB
   Image 1:  2816174 Bytes =  2.7 MB
   Verifying Checksum ... OK
=>
```

On a VCMA9 you have to use the command nand write.jffs2 to write the image to the nand flash. Refer to chapter 3.3.29 nand - NAND sub-system for more information:

```
=> nand write.jffs2 33000000 0 $filesize
NAND write: device 0 offset 0, size 778465 ... 778465 bytes write: OK
=>
```

3.5.2.4 Booting the flashed kernel

Prior to booting the kernel, you may have to change/set the bootargs:
Example PIP405:

```
=> setenv bootargs video=ctfb:mode:0,depth:8, root=/dev/ram0 ide=nodma
=> printenv bootargs
bootargs=console= video=ctfb:mode:0,depth:8, root=/dev/ram0 ide=nodma
=> saveenv
```

On a MIP405/PIP405 you can directly issue the bootm command to boot the image:

```
=> bootm ffc00000
## Booting image at ffc00000 ...
   Image Name:   kernel and initrd for MIP405
   Created:      2002-04-16  13:17:43 UTC
   Image Type:   PowerPC Linux Multi-File Image (gzip compressed)
   Data Size:    3613150 Bytes =  3.4 MB
   Load Address: 00000000
   Entry Point:  00000000
   Contents:
   Verifying Checksum ... OK
   Uncompressing Multi-File Image ... OK
   Loading Ramdisk to 07cd6000, end 07f858ae ... OK
Linux version 2.4.18-rc1-yard-boot (root@m05denis) (gcc version 2.95.3 20010315 (r
elease)) #3 Thu Apr 11 11:23:27 Local time zone must be set--see zic manual page 2
```

On a VCMA9 you must invoke the command nboot. See also chapter 3.3.30 nboot - boot from NAND device.

```
=> nboot 33000000 0
Loading from device 0: <NULL> at 0x4E000000 (offset 0x0)
Image Name: Linux-2.4.19-rmk4-swl6-dm1
Created: 2003-04-17 14:49:55 UTC
Image Type: ARM Linux Kernel Image (gzip compressed)
Data Size: 778401 Bytes = 760.2 kB
Load Address: 30008000
Entry Point: 30008000
=>
```

Then the kernel can be started with bootm:

```
=> bootm 33000000
```

3.5.3 Console assignment

The default console is the RS232 interface. On some boards alternatively in/output devices can be used, for example:
PIP405:

- Onboard keyboard controller
- USB keyboard
- Onboard VGA

MIP405:

- USB keyboard
- PC/104-Plus VGA card (currently only ct69000 boards)

VCMA9:

- Onboard LCD controller (not yet implemented)
- USB keyboard (not yet implemented)

3.5.3.1 Assign alternate console

First connect the In/Output devices on your board and start-up. If using a USB keyboard start the USB sub-system:

```
=> usb start
(Re)start USB...
USB:   scanning bus for devices... 4 USB Devices found
=>
```

With the coninfo command you can see all Input/Output Devices:

```
=> coninfo
List of available devices:
vga       80000002 S.O
serial    80000003 SIO stdin stdout stderr
kbd       80000001 SI.
usbkbd    80000001 SI.
=>
```

To assign another device to the console, you have to set environment variables:

```
=> setenv stdin kbd
=> saveenv
Saving Environment to EEPROM...
=> coninfo
List of available devices:
vga       80000002 S.O
serial    80000003 SIO stdout stderr
kbd       80000001 SI. stdin
usbkbd    80000001 SI.
=>
```

Note: The new assignment takes place immediately. So in the example above you have to enter the saveenv and coninfo command on the connected keyboard instead on the terminal.

Since the environment variables are saved, the next time the board starts-up, the devices are still assigned. In case a device is not found on start-up, the U-Boot switches automatically back to the default console assignment (that is serial line).
If you want to switch back to the default console without setting the environment variables, a DIP switch exists where you can “over write” the settings. Please refer your board documentation for information over this feature.

3.5.3.2 Working with an USB Keyboard

To use an USB keyboard you have to start the USB Sub-system first. To start this automatically on start-up, you have to set the environment variable preboot:

```
=> setenv preboot usb start
=> setenv stdin usbkbd
=> saveenv
Saving Environment to EEPROM...
=>
```

Warning: Please make sure to stop the USB subsystem before booting an OS. This can be done by adding the command `usb stop` to the variable bootcmd:

```
=> setenv bootcmd usb stop\; diskboot 400000 0:1\; bootm
=> saveenv
Saving Environment to EEPROM...
=> printenv bootcmd
bootcmd=usb stop; diskboot 400000 0:1; bootm
=>
```

© 2003 by MPL AG

3.5.4 Creating bootable kernel images

This chapter explains how a kernel binary is converted into a u-boot image. If you are looking for information how to build a kernel, refer to the kernel documentation.
The main purpose of u-boot is to load data into the memory and perform the appropriate action with this data. In case of a kernel, the action is to write it to the specified Memory location passing parameters to it and execute it.
To distinguish different data types, actions etc. the tool `mkimage` exists.

3.5.4.1 U-Boot images

U-Boot images are created with the tool `mkimage`. This is a brief introduction of the tool `mkimage`, which is a part of the u-boot project. The source of the u-boot is available at http://u-boot.sourceforge.net/. For in-deep information please refer to the README/doc and source of the U-Boot.
The tool `mkimage` adds a header to the data containing the necessary information for the u-boot:

- Architecture:
  In case of the MIP405 / PIP405 this is `ppc` in case of the VCMA9 this is `arm`.
- Operating system:
  In case of Linux this is `linux` in case of a U-Boot update it is `u-boot`. Please refer to the source/README for more Operating systems.
- Image Type:
  Following Image types currently exists: `standalone, kernel, ramdisk, multi, firmware` and `script`.
- Compression:
  `none` and `gzip`.
- Load address
- Entry point
- Image Name (max 32 characters)

To ensure that the data as well as the header information are complete an correct loaded, two CRC checksums are also added: the header CRC32 checksum and the data CRC32 checksum.

3.5.4.2 Using mkimage

Invoking `mkimage` without a parameter displays a brief help:

```
Usage: tools/mkimage -l image
          -l ==> list image header information
       tools/mkimage -A arch -O os -T type -C comp -a addr -e ep -n name -d data_file[:data_file...]
image
          -A ==> set architecture to 'arch'
          -O ==> set operating system to 'os'
          -T ==> set image type to 'type'
          -C ==> set compression type 'comp'
          -a ==> set load address to 'addr' (hex)
          -e ==> set entry point to 'ep' (hex)
          -n ==> set image name to 'name'
          -d ==> use image data from 'datafile'
          -x ==> set XIP (execute in place)
```

Invoking `mkimage` with the argument `-l` and an image file lists information about the image.
If you intend to build a linux kernel the parameters for the `mkimage` are already included in the Makefile. You have only to make sure that the make knows where the `mkimage` is located:

```
ln -s <location where mkimage resides>/mkimage /usr/bin/mkimage
```

Provided that you are using the corresponding kernel sources the u-boot image will be created when issuing:

```
make uImage
```

3.5.5 Creating a new U-Boot Image

Provided that you have already build a appropriate toolchain use following steps to build a new u-boot:

- Download the latest u-boot sources from www.u-boot.sourceforge.net
- Set the environment variable `CROSS_COMPILE` accordingly to your toolchain and make sure that the path to your cross toolchain is included in your `PATH` variable. Example:

```
export CROSS_COMPILE=powerpc-linux-
```

- Untar the sources to a <user defined directory>

```
<user defined directory>tar xjvf <downloaded u-boot path>/u-boot.tar.bz2
```

- Configure u-boot:

```
cd <user defined directory>/u-boot
make MIP405_config
```

For MPL boards following config strings are used:

MIP405	`make MIP405_config`
MIP405T	`make MIP405T_config`
PIP405	`make PIP405_config`
VCMA9	`make VCMA9_config`

- Build u-boot:

```
make
```

After successful building, the binary `u-boot.bin` should reside in the <user defined directory>.

- Create the image:

For MIP405, MIP405T and PIP405 use following command to create the u-boot image:

```
tools/mkimage -A ppc -O u-boot -T firmware -C none -a fff80000 -e 0 -n 'U-Boot for MIP405' \
-d u-boot.bin u-boot.img
```

Alternatively you can gzip the binary image which results in faster load time. This may be useful if you intend to load the image via the serial line:

```
cp u-boot.img u-boot.zip
gzip -f u-boot.zip
tools/mkimage -A ppc -O u-boot -T firmware -C gzip -a fff80000 -e 0 -n 'U-Boot for MIP405' \
-d u-boot.zip u-boot.img
```

For the VCMA9 use following command:

```
tools/mkimage -A arm -O u-boot -T firmware -C none -a 0 -e 0 -n 'U-Boot for VCMA9'
-d u-boot.bin u-boot.img
```

To update the u-boot on the target board refer to chapter 3.5.1 Updating the bootloader.

© 2003 by MPL AG

4. Support Information

4.1 U-Boot project

The U-Boot project has a mailing list which is available at www.u-boot.sourceforge.net.

4.2 MPL AG

In case of questions contact MPL AG or your local distributor.

MPL AG homepage: www.mpl.ch
Email address: support@mpl.ch

COPYRIGHT AND REVISION HISTORY

Copyright © 2003 by MPL AG Elektronikunternehmen. All rights reserved. Reproduction of this document in part or whole, by any means is prohibited, without written permission from MPL AG Elektronikunternehmen.

This manual reflects U-Boot 0.4.0.

DISCLAIMER

The information contained herein is believed to be accurate as of the date of this publication, however, MPL AG will not be liable for any damages, including indirect or consequential, arising out of the application or use of any product, circuit or software described herein.

MPL AG reserves the right to make changes to any product herein to improve reliability, function or design.

TRADEMARKS

Brand or product names are trademarks and registered trademarks of their respective holders.

Our local distributor: